Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TRABAJO DE FINAL DE CARRERA

TITULO DEL TFG: MeteorJS: Un framework full-stack para la creación de aplicaciones web real-time.

TITULACIÓN: Grau en Ingeniería Telemática

AUTOR: Jorge Mahecha Durango

DIRECTOR: Roc Meseguer

FECHA: 6 de febrero del 2017

Título: MeteorJS: Un framework full-stack para la creación de aplicaciones web real-time.

Autor: Jorge Mahecha Durango

Director: Roc Meseguer

Fecha: 6 de febrero del 2017

Resumen

En la actualidad la mayoría de aplicaciones webs están basadas en tecnologías clásicas y muy robustas como pueden ser PHP, JAVA, Python o .NET. Estas tecnologías ofrecen crear aplicaciones web potentes con frameworks muy completos pero que no están adaptados a la nueva era dónde es crucial acceder a la información y realizar acciones de una forma muy rápida. Es en este punto dónde aparecen nuevas tecnologías, como es Node.js, JavaScript y MeteorJS, que ofrecen crear aplicaciones web flexibles y escalables de una forma sencilla y con mayor control, ya que tienden a separar la interfaz del usuario de los datos. Por otra parte, son tecnologías que están pensadas para ser reactivas, es decir que reaccionan automáticamente a cualquier cambio que se produzca en la base de datos, variables o cambios ejecutados por el usuario, de forma que los datos mostrados siempre están actualizados.

El objetivo de este proyecto es evaluar el potencial de MeteorJS, ya que ofrece todas las herramientas para realizar una aplicación web de una forma fácil y de rápido aprendizaje. Al ser un framework full-stack se puede desarrollar toda la estructura de la aplicación desde la parte del servidor hasta la del cliente y haciendo uso de un solo lenguaje de programación, que es JavaScript. Entre las ventajas de desarrollar una aplicación con MeteorJS, es que ofrece un repositorio de packages, que permite integrar funcionalidades a la aplicación sin tener que desarrollarlas, ya que ya han sido desarrolladas por la comunidad, lo que implica reducir el tiempo dedicado a la aplicación y poder centrarse realmente en las funcionalidades específicas. Un ejemplo de esto puede ser la funcionalidad de registrarse o iniciar sesión en la aplicación que durante el proyecto se demostrará y a la vez se harán más demostraciones para evaluar este nuevo framework.

El proyecto se ha realizado de forma experimental en el cual se han realizado diferentes demostraciones para evaluar el potencial del framework. Cada una de las demostraciones contiene una pequeña introducción en la que se explica los objetivos que se pretenden conseguir con cada una de ellas, se mostrarán los pasos y las partes de código más importantes que se han utilizado para conseguir los objetivos y terminar con las conclusiones realizadas con la demostración.

Una vez que las demostraciones se han completado, se realizará la parte final del proyecto, que consiste en una aplicación web uniendo todas las demostraciones anteriores y con lo que se conseguirá una aplicación con diferentes secciones. Estas secciones consisten en una página principal que será el Homepage de la aplicación, una sección para el feedback o comentarios, y un menú diferente en el caso de que el usuario este registrado. En el caso de que el usuario este registrado será capaz de crear y editar estudiantes, crear e invitar al resto de estudiantes a una sala de conversación o unirse a las que haya sido invitado y de esta forma mantener una conversación en tiempo real con los usuarios que estén en línea.

Title: MeteorJS: Un framework full-stack para la creación de aplicaciones web real-time.

Author: Jorge Mahecha Durango

Director: Roc Meseguer

Date: 6 de Febrero del 2017

Overview

Nowadays mostly all the web apps are build using classic and robust technologies such as PHP, JAVA, Python o .NET. These technologies are able to offer powerful frameworks but it is not adapted to the new times where is critical to access on the information and do actions as fast as possible. In this point is where are appearing new technologies such as Node.js, JavaScript and MeteorJS that are offering build flexible and scalable web apps without make the code too complex and always having the control over the app as the data and UI (user interface) are separated. By other hand, these technologies are built to be reactive so any change that has been done in the database or variables are updated automatically.

The objective of this project is to evaluate the potential of MeteorJS, since it offers all the tools to make a web application in a simple and quick learning. MeteorJS is a full-stack framework that means that you can develop the entire structure of the application from the server side to the client and making use of a single programming language, which is JavaScript. Among the advantages of developing an application with MeteorJS, is that it offers a package repository, which allows integrating functionalities to the application without having to develop them because of they have already been developed by the community, which implies reducing the time dedicated to the application and can really focus on specific functionalities. An example of this can be the functionality of registering or logging into the application that will be demonstrated during the project and at the same time more demonstrations will be done to evaluate this new framework.

The project has been carried out experimentally in which different demonstrations have been carried out to evaluate the potential of the framework. Each of the demonstrations contains a short introduction explaining the objectives to be achieved with each of them, will show the most important steps and pieces of code that have been used to achieve the objectives and finishing with the conclusions about the demonstrations.

Once the demonstrations have been completed, the final part of the project will be creating a web application joining all the previous demonstrations and splitting it in different sections. These sections consist of a main page that will be the homepage of the application, a section for feedback, and showing a different menu in case the user is registered. In case the user is registered will

be able to create and edit students, create and invite other students to a meeting room or chat room, or join to the meeting room which he has been invited so we will be able to have a real-time conversation with users who are online.

ÍNDICE

CAPITULO 1. INTRODUCCIÓN

La realización de este proyecto consiste en evaluar el potencial y las características más importantes del framework Meteor.js, el cual es un framework web, full-stack para Node.js, gratuito y se distribuye de forma libre. El primer lanzamiento del framework con el nombre actual fue en octubre del 2014 por lo que es un framework reciente, pero aun así se puede encontrar mucha documentación en las redes.

Al ser un framework web full-stack nos proporciona todas las herramientas necesarias para el desarrollo de una aplicación web, tanto en el lado del servidor como en el del cliente, de manera que nos permite desarrollar una aplicación de forma rápida, moderna y de fácil aprendizaje.

Las motivaciones principales que llevaron a la realización del proyecto es demostrar que se pueden crear aplicaciones web si tener que preocuparse en las conexiones con la base de datos ni llamadas AJAX al servidor y siempre mostrando cualquier cambio que se realice en la base de datos en real time y de forma automática. Además de demostrar que con las herramientas que ofrece MeteorJS se reduce el tiempo a la hora de crear una aplicación.

El propósito del proyecto es realizar diferentes demostraciones en las cuales se puedan ver las ventajas y el potencial de utilizar MeteorJS y finalizando con una demostración final en la que englobe las demostraciones anteriores.

1.1 Motivación del proyecto

En la actualidad la mayoría de aplicaciones webs están basadas en tecnologías clásicas y muy robustas como pueden ser PHP, JAVA, Python o .NET. Estas tecnologías ofrecen crear aplicaciones web potentes y robustas con poderosos frameworks pero que a la misma vez no se adecuan a las necesidades de algunas aplicaciones actuales, en las que la rapidez y la información en tiempo real es crucial.

Por otro lado, limitan el tiempo que el programador le dedica realmente a las funcionalidades de la aplicación de la cual quiera crear ya que debe tener en cuenta también las comunicaciones entre cliente, servidor y base de datos.

En el escenario actual hay una nueva tecnología que es Node.js, la cual permite crear aplicaciones web basadas solo en JavaScript, tanto en el cliente como el servidor y en la que uno de los frameworks más útiles es MeteorJS ya que proporciona todas las herramientas para superar las limitaciones anteriores y dando al programador la sensación de que toda la información está disponible para él y por lo tanto reduciendo el tiempo de programación.

Usualmente estos frameworks se adaptan y adoptan a nuevos frameworks mucho más rápido que las tecnologías clásicas. Un ejemplo de ello es que, durante la realización del proyecto, MeteorJS ha pasado de solo trabajar con MongoDB a ser compatible con cualquier tipo de base de datos gracias a Apollo Server[1] que se basa en GraphQL[2] desarrollado por Facebook.

Por lo que además de las motivaciones mencionadas anteriormente, la realización de este proyecto quiere evaluar el potencial a la hora de crear aplicaciones web usando Node.js como servidor y MeteorJS como framework.

1.2 Estructura del proyecto

La estructura del proyecto consistirá en la introducción a MeteorJS en la cual se explicarán los componentes que forman el framework, cómo el tipo de base de datos que utiliza y en la forma en la que se puede acceder a la información guardada en este tipo de base de datos. Ya que durante el proyecto se utilizará estos principios básicos para recuperar información guardada en la base de datos. Por otro lado, también se explicará como se realiza la comunicación entre cliente y servidor que nos ayudará a entender como es actualizada la información que esta guardada en el servidor y que es actualizada automáticamente en el cliente. Finalmente se comentarán los puntos a destacar y de las ventajas de utilizar MeteorJS como framework a la hora de crear aplicaciones web.

Una vez se ha realizado la introducción a MeteorJS se explicarán las demostraciones en la cual cada una contiene una pequeña introducción sobre la que se explica los objetivos que se pretenden conseguir con cada una de ellas, se mostrarán los pasos y las partes de código más importantes que se han utilizado para conseguir los objetivos y terminar con las conclusiones realizadas con la demostración.

Al terminar las demostraciones especificas se realizará una aplicación web en la cual engloba todas las demostraciones que se han realizado anteriormente, y en la que habrá una demostración en vivo el día de la presentación. En el siguiente punto de objetivos del proyecto se hablará con más detalle sobre esta aplicación final.

Finalmente se hablarán de las conclusiones que se hayan realizado durante el proyecto y la evaluación de MeteorJS como framework para desarrollar aplicaciones web, además de las ventajas que conlleva utilizar este tipo de framework. Por otro lado se comentarán los inconvenientes que se podrían encontrar a la hora de utilizar MeteorJS que debido al ser un framework relativamente nuevo y por lo que no es tan robusto cómo pueden ser el caso de otros frameworks. También se comentarán las conclusiones personales y las recomendaciones para una persona que desee realizar un proyecto real utilizando MeteorJS.

---

[1] Apollo Server: Para conocer más sobre Apollo Server se puede visitar apollodata.com
[2] GraphQL: Más información sobre GrapQL en graphql.org

1.3 Objetivos del proyecto

Una vez las demostraciones se han completado, se realizará la parte final del proyecto, que consiste en una aplicación web uniendo todas las demostraciones anteriores y con lo que se conseguirá una aplicación con diferentes secciones. Estas secciones consisten en una página principal que será el Homepage de la aplicación, una sección para el feedback o comentarios, y un menú diferente en el caso de que el usuario este registrado. En el caso de que el usuario este registrado será capaz de crear y editar estudiantes, crear e invitar al resto de estudiantes a una sala de meeting o de chat, o unirse a la sala de meeting a las que haya sido invitado y de esta forma mantener una conversación en tiempo real con los usuarios que estén en línea.

De esta manera se podrá evaluar el potencial de MeteorJS a la hora de crear aplicaciones web y que servirán para realizar un proyecto real, de una forma sencilla y rápida en la que el programador se puede centrar en las funcionalidades específicas de la aplicación.

1.4 Objetivos específicos

Los siguientes puntos tienen la finalidad de demostrar y evaluar técnicamente que utilizar MeteorJS es una herramienta que agiliza el desarrollo de aplicaciones y el cual ofrece todas las herramientas para crear una aplicación desde cero hasta conseguir una aplicación totalmente funcional, rápida y moderna. A continuación, se listarán estos puntos:

1. Se realizarán diferentes ejemplos para demostrar que, con la ayuda de los packages, se puede aumentar la velocidad de programación y reducir el número posible de bugs o errores en el código.
2. Se realizarán ejemplos para demostrar la creación de rutas dinámicas, y la creación de enlaces automáticos independientemente de la ubicación de los archivos, lo que permite una mejor organización del proyecto.
3. Se realizarán diferentes ejemplos para demostrar la gestión de eventos accionados desde el navegador y cómo se puede acceder a la base de datos sin la necesidad de crear conexiones previas con el servidor para recuperar la información.
4. Se realizarán diferentes ejemplos para demostrar la sincronización de los datos en real-time sin tener que crear funciones específicas para esto, ya que el mismo framework es el que se hace cargo de actualizar los cambios automáticamente.
5. Finalmente se demostrará que MeteorJS es un framework con mucho potencial ya que permite desarrollar una aplicación web de una forma rápida, moderna y a la vez flexible que permite añadir cualquier funcionalidad de una forma rápida y todo esto utilizando sólo un lenguaje de programación

CAPITULO 2. MeteorJS

A continuación, se hará paso a la introducción de MeteorJS, en la cual se hablará sobre los componentes que lo forman tanto en el servidor como en el cliente, el tipo de base de datos utilizada en el proyecto y las diferencias que hay con MySQL. Esta explicación sobre la base de datos servirá de principios básicos durante el proyecto para insertar, actualizar y eliminar información de la base de datos. Por otro lado, se explicará el protocolo utilizado para la comunicación entre cliente y servidor, el cual proporciona la ventaja de no tener que realizar llamadas al servidor ya que se realizan de forma automática. Finalmente se nombrarán los puntos a destacar y las ventajas de utilizar MeteorJS en cualquier proyecto.

2.1 Arquitectura de Meteor.js

La arquitectura de Meteor.js en este proyecto la reflejaremos en la siguiente imagen y a partir de la cual se explicarán las partes más importantes:

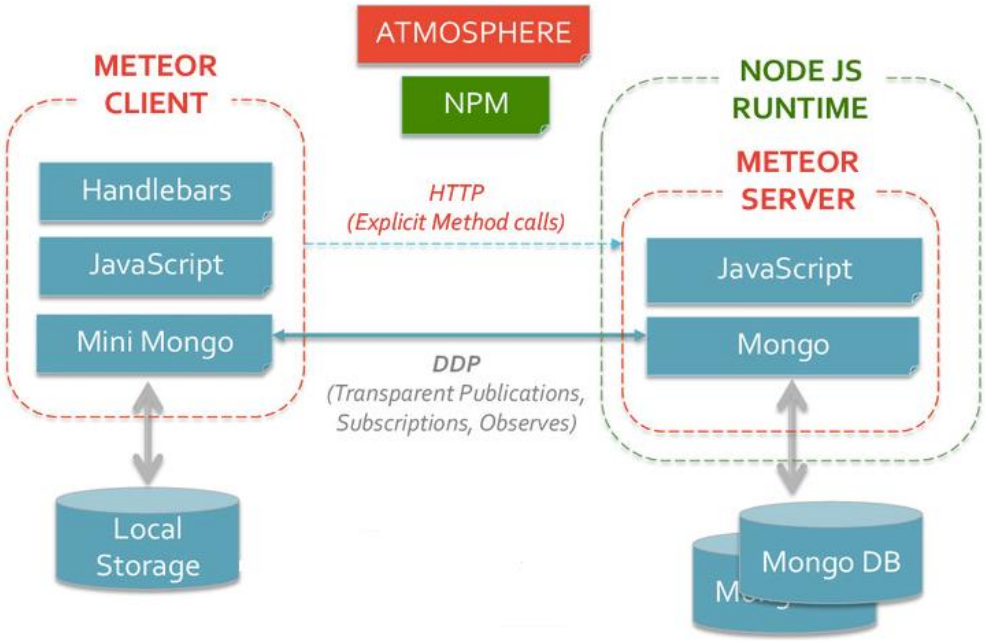

Fig. 1 Arquitectura de Meteor.js

Como se puede apreciar en la imagen anterior en la parte del cliente tendremos Blaze (Handlebars) que es un lenguaje de plantillas optimizado para MeteorJS e inspirado en los Handlebars[3] el cual permite crear plantillas reactivas o es decir cualquier cambio que se produzcan en las variables de la plantilla será reflejado automáticamente.

[3] Handlebars. En el siguiente enlace se puede encontrar más información : handlebarsjs.com

La utilización de handlebars permite introducir variables dentro de las plantillas, cómo sería el caso de introducir código PHP dentro del HTML, pero con la diferencia que los cambios que se realicen en las variables de los handlebars serán actualizados automáticamente en la plantilla a diferencia de PHP que genera código estático en el HTML.

Por otro lado, también en el cliente se tiene una versión de la base de datos del servidor llamada Mini Mongo, que consiste en un objeto JSON el cual tiene los datos de la base de datos y que permite la sincronización con la base de datos del servidor. Al tener una mini réplica de la base de datos en el cliente permite tener controlado los datos que se manejaran en las funciones y de la cual se puede acceder a través de la consola del navegador, por lo que proporciona una herramienta muy útil a la hora de realizar seguimiento de las variables.

El protocolo utilizado para la sincronización entre cliente y servidor se llama DDP (Distributed Data Protocol) y esta diseñado por MeteorJS para ofrecer la característica de mantener las dos bases de datos del cliente y el servidor sincronizada con los últimos datos.

A continuación, se explicarán con más detalles las partes mencionadas anteriormente.

2.2 MongoDB, colecciones y Meteor.js

MongoDB es una base de datos NoSQL, por lo que no tiene una estructura basada en la relación entre tablas, sino que utiliza el formato JSON para guardar la información, lo que permite flexibilidad a la hora de guardar los datos.

En MySQL la relación entre tablas se realizaría con foreign keys o realizando uniones entre tablas, por ejemplo, si tenemos dos tablas con las siguientes informaciones:

Students		**Subjects**		
ID	Name	ID	Name_subject	Student_id
1	Manuel	1	Matematicas	1
2	Pedro	2	React.js	1
3	Cris	3	Fisica	2

En MySQL se podría relacionar las tablas de la siguiente forma:

```
SELECT * FROM students LEFT JOIN subjects ON students.id = subjects.student_id
```

En MongoDB se tendría una sola colección con los siguientes datos:

```
{
students: {
        _id: 1 , name: Manuel , subjects: [
                                                {id: 1 , name_subject: Matematicas},
                                                {id: 2 , name_subject: React.js}
                                            ] ,
        _id: 2, name: Pedro, subjects:   [
                                                {id: 3 , name_subject: Fisica }
                                         ]
 }
}
```

Es decir, no es necesario realizar relaciones entre tablas en MongoDB ya que todos los datos se pueden organizar en un solo formato JSON. Este formato permite tener bases de datos muy flexibles ya que se pueden añadir nuevos campos en las entradas que sea necesario que por el contrario en MySQL esto no sería posible ya que se debería introducir una nueva columna o campo para todos los registros, lo que hace MySQL menos flexible.

La estructura de una base de datos en MongoDB y su relación en MySQL es la siguiente:

MongoDB	MySQL
Database	Database
Collection	Table
Document	Row
Field	Value
Primary Key	Primary Key

Tabla 1 Relación MongoDB y MySQL

Para dar una idea la siguiente imagen muestra lo que sería un Document en MongoDB que sería el equivalente a una fila o registro en MySQL, donde cada valor sería una columna o campo.

Fig. 2 Document en MongoDB

MeteorJS proporciona todas las herramientas para poder acceder y manipular la información en MongoDB sin tener que crear una conexión con la base de datos cada vez que queramos utilizarla.

En el caso de querer crear una nueva colección, insertar un nuevo documento o buscar en ellos se puede realizar de la siguiente forma:

1) **Crear Colección**: PlayersList = new Mongo.Collection('players');

En este caso se creará una colección llamada **“players”** y será guardada en la variable **PlayersList**. Esta variable permitirá acceder a la colección para realizar cualquier acción que se desee sobre ella. El equivalente a MySQL de este ejemplo sería crear una nueva tabla llamada ‘players’, pero que a diferencia de MongoDB, se tendría que definir los campos o columnas que dispondrá esta tabla.

2) **Insertar Document**: PlayersList.insert({ name: "David", score: 0 });

Para insertar un documento en la colección simplemente es añadir los campos que se quieran utilizando la variable que se ha definido anteriormente. Para el caso de MySQL el equivalente sería realizar un INSERT de la siguiente forma:

INSERT INTO players (name, score) VALUES (“David”, 0);

3) **Buscar**: PlayersList.find({ name: "David" }).fetch();

Finalmente para realizar una búsqueda en la colección se debe seleccionar el campo por el que se quiere buscar, que en este caso es cualquier registro que contenga el nombre “David”. El equivalente en MySQL seria realizar un SELECT de la siguiente forma:

SELECT * FROM players WHERE name = “David”

Estos conceptos serán muy importantes durante el proyecto ya que son los que se utilizarán para recuperar o modificar datos en las demostraciones y en la aplicación final.

2.3 Estructura

MeteorJS permite tener una estructura de código muy flexible, ya que define carpetas dónde irán los archivos para el cliente, servidor, compartidos o públicos. Una vez estos estos archivos están dentro de cualquiera de estas carpetas pueden ser accedidos fácilmente y cómo se verá más adelante con el enrutamiento dinámico, la ubicación del archivo no es tan importante como lo es el nombre de la plantilla.

La forma básica de estructurar el proyecto es la siguiente:

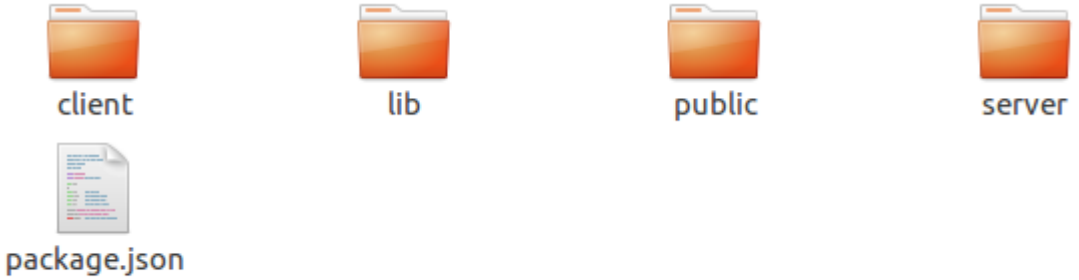

Fig. 3 Estructura del proyecto

Como se puede ver hay una carpeta llamada client y server. En la carpeta client estarán todos los archivos relacionados con el cliente y con el servidor respectivamente. Por lo que las variables que estén definidas en el servidor no serán accesibles desde el cliente, ya que por motivos de seguridad no todas las variables pueden ser accedidas desde el cliente ya que pondría en riesgo datos críticos de la aplicación, como pueden ser el usuario y la contraseña de la base de datos.

La carpeta **lib** contiene los archivos compartidos entre servidor y cliente, lo que permite compartir variables entre ellos y es el punto de unión entre ambos lados de la aplicación. La carpeta public contiene los archivos que son globalmente accesibles, como imágenes, documentos o más archivos que queramos compartir.

Finalmente el archivo package.json proporciona información sobre el proyecto y además nos permite añadir packages o herramientas a nuestro proyecto de forma manual.

2.4 Sincronización entre cliente y servidor

MeteorJS utiliza el protocolo DDP (Distributed Data Protocol) para las comunicaciones entre servidor y cliente, por lo que la utilización de técnicas como AJAX no es necesaria. Cualquier cambio que se produzca en la información que se comparte con el cliente será actualizado de forma automática.

La forma de trabajar del protocolo DDP es la siguiente:

1. En el lado del servidor, se debe publicar la información que queramos que esté disponible desde el lado del cliente. Esto se realiza utilizando el siguiente método: Meteor.publish.(**name_collection**)
2. En el lado del cliente se debe suscribir a la colección que ha sido publicada por el servidor, utilizando un método similar al anterior, Meteor.subscribe.(**name_collection**)

Una vez se realiza la subscripción, el protocolo DDP realiza la sincronización en la base de datos del cliente, Mini Mongo, y la base de datos del servidor Mongo DB, por lo que el cliente siempre estará accediendo a información actualizada y lo que facilita mucho el trabajo del programador ya que no tiene que desarrollar ninguna función para estar actualizando estos datos.

Este es uno de los puntos claves de MeteorJS y de las cuales proporciona una gran ventaja a la hora de realizar aplicaciones web con este framework.

2.5 Puntos a destacar

A continuación, se enumerarán las ventajas y los puntos a destacar a la hora de utilizar un framework como MeteorJS:

1. **La rapidez al desarrollar una aplicación web**. Hay infinidades de herramientas (packages) que se pueden instalar en nuestro proyecto haciendo que muchas veces se pueda ahorrar tiempo y código por la ayuda de estos packages. Un ejemplo de esta característica es poder generar formularios automáticamente basados en nuestra base de datos, en la que podamos incluir validación y la personalización que queramos. Más adelante veremos la demostración de este punto con más ejemplos.

2. **Desarrollo con un solo lenguaje**. Javascript es uno de los lenguajes más populares en la actualidad y de fácil aprendizaje, lo que permite una sencilla adaptación a Meteor.js.

3. **Información en Real-time**. La estructura nativa de MeteorJS es ofrecer información en Real-time, por lo que cualquier cambio que se produzca en las bases de datos o las variables proporcionadas en las plantillas se reflejará automáticamente en la página web sin tener que volver a cargar la página. Ideal cuando la información varía constantemente.

4. **Aplicaciones web de una sola página**. Al utilizar enrutamiento dinámico cada vez que se realiza un cambio de página o de enlace, solo se refresca el contenido variable, por lo que el resto permanece sin modificaciones, lo que provoca que la aplicación web cargue mucho más rápido ya que los archivos HTML, CSS y Scripts son cargados una sola vez.

5. **Más fácil realizar diseños responsives para diferentes dispositivos**. Al constar de una sola página, es mucho más facil diseñar plantillas que se vean adecuadamente en dispositivos móviles, ya que no se debe aplicar un diseño para cada una de las páginas.

6. **Separación de los datos y de la interfaz del usuario.** Al tener los datos y las funciones separadas de la interfaz del usuario hace que la estructura del código sea más organizado y que permita añadir nuevas páginas o funcionalidades de forma muy flexible y escalable.

CAPITULO 3. Posibilidades Tecnológicas de Meteor.js

En este capítulo se centrará en las posibilidades tecnologías de Meteor.js y se harán a través de tres demostraciones que empezarán por lo más básico como es insertar y modificar datos en la base de datos a través de packages y a la vez se verá la utilidad de otros packages que también serán utilizados durante las demostraciones y sus beneficios.

Como segunda demostración se crearán diferentes páginas para mostrar el enrutamiento dinámico y sus ventajas y finalmente el uso de eventos en Meteor.js para interactuar con la base de datos dependiendo de los eventos realizados por el cliente.

En resumen, estos serán los tres puntos que se demostrarán:

1. **Utilización de packages**: Al final de la demostración se verá que se ahorra mucho tiempo programando una aplicación con el uso de packages y a la vez se disminuye el número de bugs ya que son packages que son utilizados por una gran parte de la comunidad.
2. **Enrutamiento dinámico**: Se verá que el enrutamiento se realiza programáticamente y que cada enlace generado será enlazado con la plantilla que se desee, independiente de la ubicación del archivo, como es el caso en las páginas webs clásicas.
3. **Eventos**: A través de los eventos se verá como son gestionados con Meteor.js y a la vez se demostrará que no se necesita ningún tipo de AJAX con el servidor para realizar cambios en la base de datos.

3.1 Utilización de packages para aumentar la velocidad de programación y reducir el número de bugs.

3.1.1 Introducción a los packages.

AtmosphereJS es el repositorio oficial donde se pueden encontrar todos los packages disponibles. Esta herramienta de MeteorJS es muy útil ya que permite aumentar la velocidad de programación utilizando funcionalidades que ya han sido previamente desarrolladas.

Por ejemplo, para la primera demostración que se hará se necesitará un package relacionado con los esquemas, que permita validar la información en los datos que se insertarán en MongoDB.

Para ello basta con visitar la página AtmosphereJS y buscar los packages relacionados con los esquemas. Como se puede observar en la siguiente imagen hay diferentes opciones, en este caso se utilizará el primero de la lista ya que es el más adecuado para la demostración.

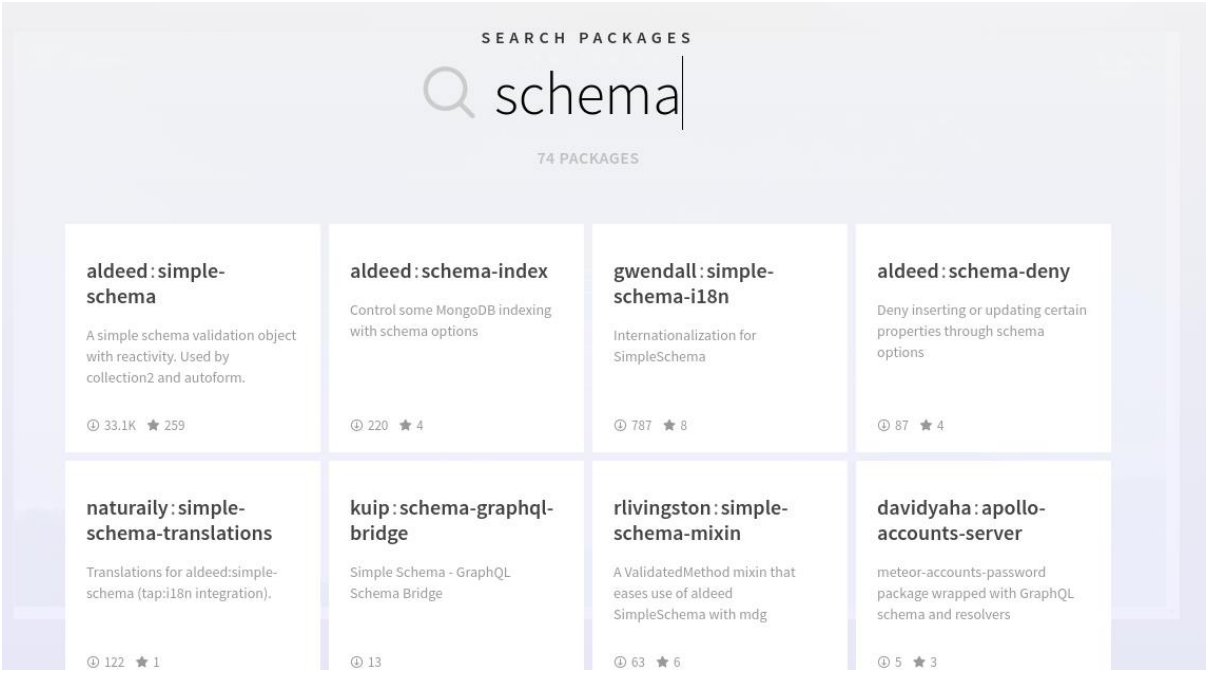

Fig. 4 Atmospherejs.com - Repositorio de packages

Cada package muestra información importante como se puede ver en la imagen siguiente:

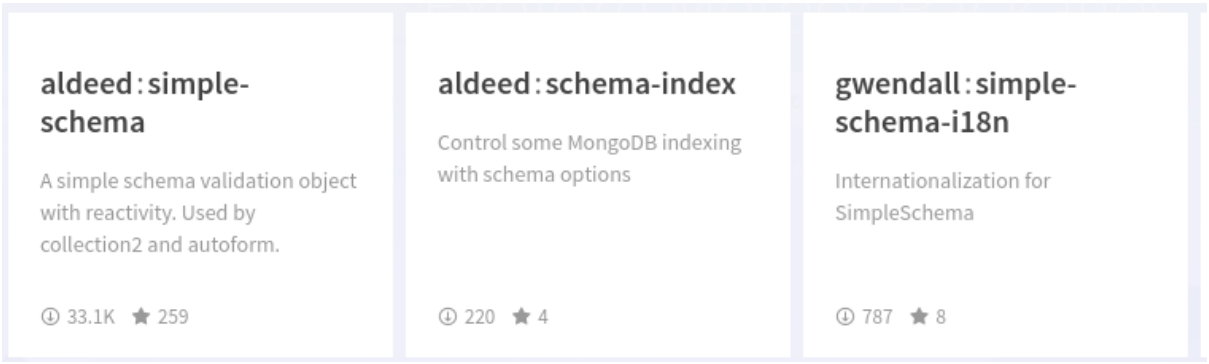

Fig. 5 Información sobre el package

En el título se puede ver el autor y nombre del package. En el primer caso el autor es aldeed y el nombre del package es simple-schema.

En la parte inferior se puede ver la descripción y el número de descargas, en el primer caso es de 33.1K descargas por lo que se puede esperar que es un package robusto y que hay muchos más programadores utilizándolo.

Finalmente, para la instalación del package se debe ejecutar el siguiente comando:

meteor add aldeed:simple-schema

3.1.2 Schemas y Autoforms

Una de las ventajas de trabajar con esquemas es que permite generar formularios que están conectados con las colecciones en MongoDB por lo cual no hay que preocuparse en la comunicación entre el cliente y la base de datos para insertar o actualizar los datos. Otra ventaja es que los formularios para crear y editar son generados automáticamente por lo que no se necesita crearlos manualmente.

Para instalar los packages simplemente es ejecutar los siguientes comandos:

1. **meteor add aldeed:simple-schema** Permite crear los esquemas, es decir los campos que tendrá la colección como también el tipo de dato que se espera, es decir si el campo es del tipo integer ,string, date etc. Por lo tanto, a la misma vez se realiza la validación de datos automáticamente.

   También da la posibilidad de tener campos con auto valores que son útiles por ejemplo en el caso de querer tener un campo con la fecha y hora en la cual se realiza la actualización de un documento, por lo que el valor puede ser obtenido automáticamente sin tener que añadirlo manualmente cada vez que se actualiza el documento.

2. **meteor add aldeed:collection2** - Permite asociar un esquema a una colección en MongoDB, es decir la estructura que tendrá cada uno de los nuevos registros que se añadan a la colección. Esta unión es la que permitirá a los auto formularios añadir la información directamente a la colección, ya que el formulario estará relacionado con el esquema y el esquema con la colección.

3. **meteor add aldeed:autoform** - Permite crear los formularios automáticamente y ofrece nuevas opciones a los esquemas nombrados en el punto número uno, ya que en el mismo esquema se puede definir cómo serán mostrados los campos en el formulario, como pueden ser las etiquetas, placeholders , si es requerido o opcional y el tipo de input que se quiera mostrar. Por ejemplo se puede seleccionar entre un input de solo texto, un desplegable con diferentes opciones, una casilla de selección etc.

El siguiente código muestra un ejemplo del esquema con una estructura del tipo JSON.

```
//Creación de la colección
Students = new Meteor.Collection('Students');

//Creación del esquema
Students.schema = new SimpleSchema ({
        name: {
                        type:String,
                        max: 60
                 },
        age:  {
                        type:Number
        },
        subjects: {
                        type:[Subjects]
        },
        createdAt: {

                type: Date,
                autoValue: function (){
                        if (this.isInsert)
                                return new Date();
                },
                autoform: {
                        type: 'hidden'
                }
        }

});
```

Cada campo tiene los siguientes valores:

1. Nombre del campo.
2. Tipo: String, Number, Date, Boolean u Object
3. Longitud del campo si es necesario.
4. Se puede definir un valor automático del campo utilizando la variable autoValue.
5. En el campo autoform se puede añadir propiedades que afectan únicamente a la generación del formulario. Por ejemplo type:hidden pondrá este campo como oculto en el formulario.

Después de haber creado el esquema simplemente es asociarlo a la colección que se desee, para esta demostración se asociará a la colección de estudiantes y se realizará de la siguiente forma:

```
Students.attachSchema(Students.schema);
```

Una vez se tiene el esquema y la colección se puede generar un formulario que permita añadir un nuevo estudiante o modificarlo. Para crear el formulario se añade la siguiente línea de código:

```
{{> quickForm collection="Students" id="insertStudent" type="insert"}}
```

El comando define tres variables que son las siguientes:

1. Collection: Nombre de la colección a la que está asociado el formulario.
2. ID : ID que se generará en el documento HTML para el formulario.
3. Type: insert o update.dependiendo del formulario que se desee.

El resultado es el formulario que se ve en la imagen inferior y que se ha realizado con **una sola línea de código** en el HTML, lo que supone un ahorro de tiempo considerable y que puede ser reutilizado cuantas veces se requiera.

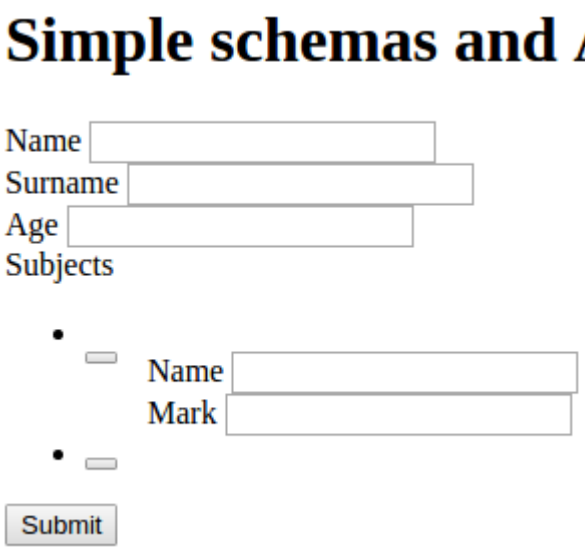

Fig. 6 Formulario creado automáticamente con el package autoform.

Como se puede observar es un formulario básico, sin ningún tipo de ***styling.*** Para añadir el styling, se puede instalar Bootstrap[4], el cual es uno de los framework más conocidos que contiene plantillas de diseño basadas en HTML y CSS. Este framework es utilizado en proyectos de la NASA y MSNBC[5].

Para instalar Bootstrap solo es necesario ejecutar el siguiente comando:

meteor add twbs:Bootstrap

La siguiente imagen muestra el formulario con Bootstrap y se puede comprobar que es un formulario mucho más amigable que en el caso anterior.

[4] Bootstrap: es.wikipedia.org/wiki/Twitter_Bootstrap
[5] MSNBC: es.wikipedia.org/wiki/MSNBC

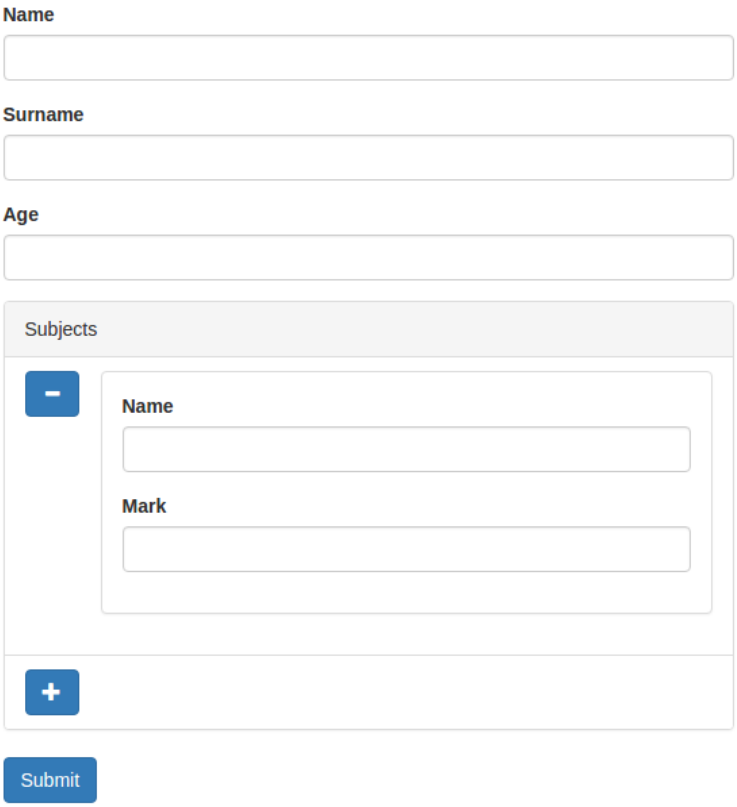

Fig. 7 Formulario con Bootstrap

Para verificar el funcionamiento del formulario se puede instalar el package **meteortoys:allthings** que permite ver la base de datos del cliente, es decir el Mini Mongo desde el navegador, por lo que podemos ver las nuevas entradas en la colección Students que fue creada anteriormente.

Además de permitir ver la colección también se pueden editar, añadir y eliminar registros en la base de datos desde el navegador, lo que permite un control de la base de datos casi absoluto. Cabe recordar que esta opción solo está disponible en el entorno desarrollador de MeteorJS ya que cuando la aplicación está en un entorno de producción esta opción no está disponible debido a que por motivos de seguridad sería peligroso que cualquiera pudiera acceder a esta base de datos de una forma tan sencilla.

Para la instalación del package simplemente es ejecutar el siguiente comando en el terminal:

meteor add meteortoys:allthings

Al rellenar todo el formulario se puede ver en las siguientes imágenes que se ha insertado un nuevo documento en la colección:

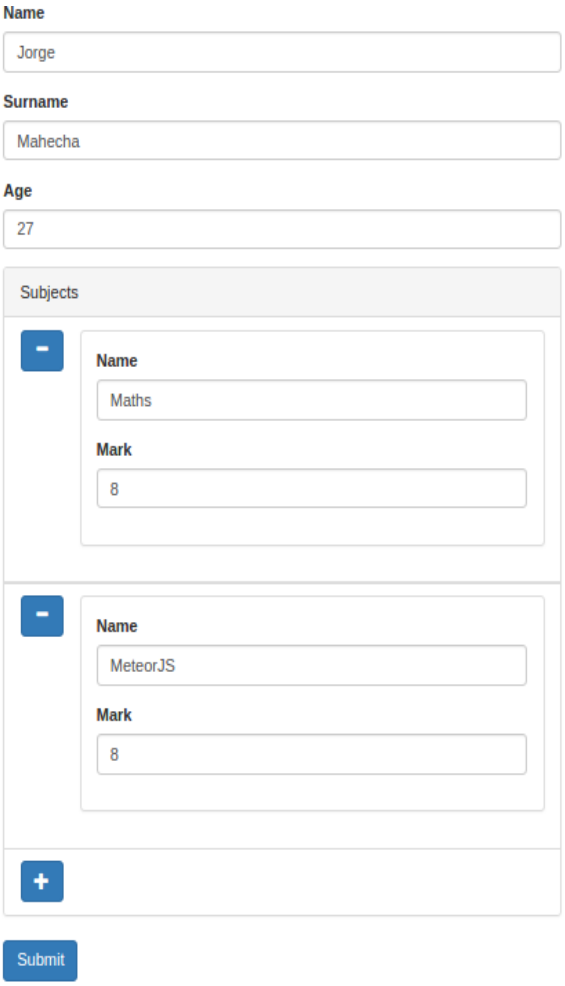

Imagen 10 Formulario

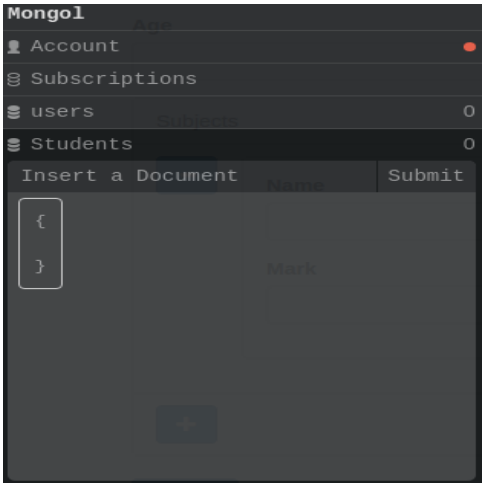

Imagen 11 Antes de rellenar el formulario

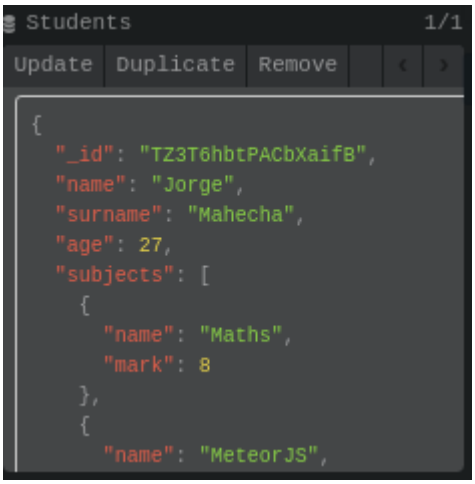

Imagen 12 Después de rellenar el formulario

En la izquierda se ve el formulario con todos los datos necesarios y en la derecha en la parte superior se puede ver una colección de Students vacía, ya que en ese momento no se había creado ningún registro. En el momento que se presione Submit se verá cómo se actualiza la colección Students con todos los datos que se han introducido. De esta forma se puede comprobar que el funcionamiento del formulario es correcto y que con el package **meteortoys** es muy fácil de detectar si hay algún tipo de problema o si la estructura de la colección no era la deseada.

Para terminar esta primera demostración con packages se puede concluir que se ha generado un formulario sincronizado con la base de datos en muy poco tiempo y que es totalmente funcional además de ser reutilizable ya que se puede utilizar en más plantillas.

3.1.3 Usuarios y cuentas

Muchas veces es necesario crear un formulario de inicio de sesión en el que normalmente se utiliza el email como nombre de usuario y una contraseña. Con la ayuda de los siguientes packages esto se convierte muy fácil de implementar:

meteor add accounts-password. Este package ofrece todas las funciones necesarias para realizar el registro y el inicio de usuario con email y contraseña además de ofrecer reiniciar la contraseña en caso necesario. Por lo que una vez instalado el package cualquier persona se podrá registrar o iniciar sesión sin tener el programador que desarrollar estas funciones.

Además, que para cada evento se pueden asociar funciones o callbacks, por ejemplo, si se requiere actualizar una variable o campo en la base de datos cada vez que se presiona el botón de iniciar sesión, se podría realizar utilizando las funciones que proporciona el mismo package.

meteor add accounts-ui. El package accounts-ui ofrece toda la Interfaz del usuario, que en este caso es una ventana dentro la aplicación web donde se puede introducir el email y la contraseña para iniciar sesión.

En la plantilla HTML simplemente es añadir la siguiente línea:

```
{{> loginButtons}}
```

```
<div class="row">
    <div class="container">
        <div class="col-md-12">

          {{> loginButtons}}

        </div>

    </div>


</div>
```

Fig. 8 Generar Formulario de inicio de sesión

De esta forma se generará la opción de iniciar sesión, crear una nueva cuenta o reiniciar la contraseña como se puede ver en las siguientes imágenes.

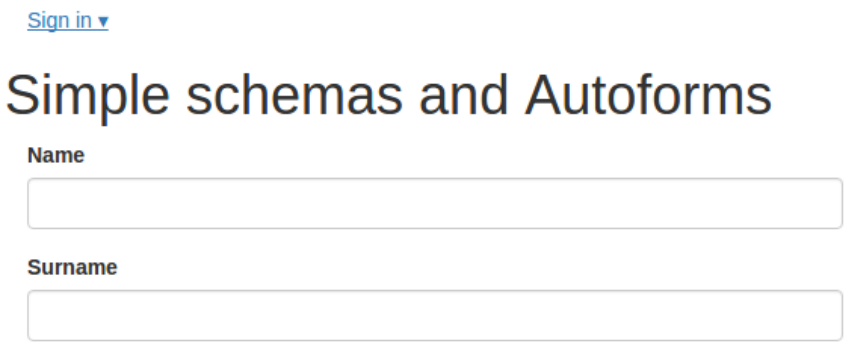

Fig. 9 Enlace para iniciar sesión

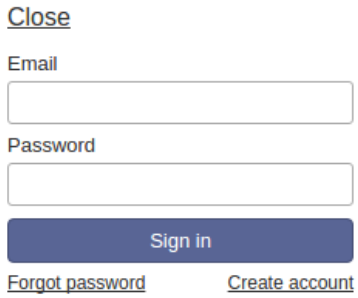

Fig. 10 Iniciar la sesión

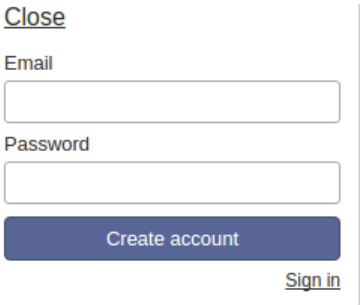

Fig. 11 Crear cuenta

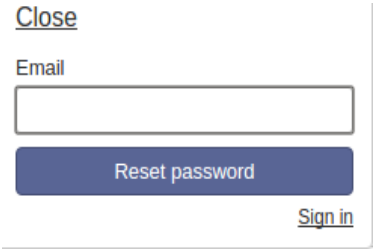

Fig. 12 Reiniciar contraseña

Cada usuario una vez es registrado es añadido automáticamente a la colección users en MongoDB. En la siguiente imagen se puede ver la estructura del usuario en la base de datos:

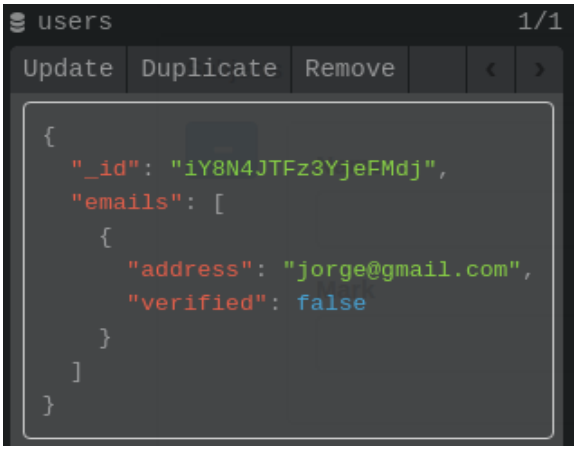

Fig. 13 Estructura de un usuario en la base de datos

Una vez el usuario ha iniciado sesión se genera automáticamente el formulario para terminar la sesión:

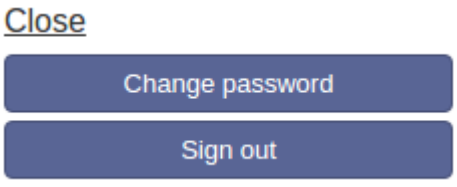

Fig. 14 Cerrar sesión

Para realizar un inicio de sesión con Facebook o con otras plataformas el procedimiento es muy similar, solo se requiere añadir uno o más de los siguientes packages:

meteor add accounts-facebook
meteor add accounts-google
meteor add accounts-github
meteor add accounts-twitter
meteor add accounts-meetup
meteor add accounts-meteor-developer

Como se ha podido ver en muy poco tiempo se puede generar una página web con inicio de sesión de una forma muy sencilla y rápida.

Si un programador necesita realizar estas funciones por su cuenta, le requeriría mucho más tiempo ya que debe desarrollar todas las funciones que se necesitan para iniciar y cerrar sesión y también crear usuarios.

El inicio de sesión es una funcionalidad necesaria en algunas aplicaciones, pero es una funcionalidad de segundo plano por lo que el uso del package es una gran ventaja para ahorrar tiempo y centrarse en las funcionalidades que realmente harán destacar a la aplicación.

3.1.4 Conclusiones sobre la primera demostración

Esta primera demostración ha permitido comprobar que la utilización de packages incrementan notablemente el rendimiento del programador ya que no necesita programar formularios para insertar o modificar datos en la base de datos, además que con la utilización de packages se pueden incluir funciones que ya han sido previamente desarrolladas como puede ser el registro de usuarios. Además, que packages como el de Bootstrap ofrecen un resultado final muy amigable.

En conclusión, de esta primera parte se puede decir que el uso de packages permite al programador centrarse en las funcionalidades que son realmente importantes para la aplicación web sin tener que dedicarse a funcionalidades que están en segundo plano.

3.2 Iron:router: Enrutamiento dinámico

3.2.1 Introducción al enrutamiento dinámico

Los proyectos MeteorJS tienen la característica que son “mono página” es decir todas las páginas son cargadas en una sola, lo que permite mucha más rapidez a la hora de cargar, ya que se cargan sólo los elementos que han cambiado.

La ventaja del enrutamiento dinámico es que permite mayor flexibilidad a la hora de organizar y gestionar el proyecto ya que los enlaces no estarán restringidos a la ubicación de los archivos, sino que por el contrario estarán relacionados con plantillas, independientemente de la ubicación.

3.2.2 Demostración básica

El package iron:router, permite crear rutas enlazando plantillas que ya han sido creadas, lo que a diferencia del routing clásico, no dependen de la ubicación del archivo, otra característica es que además ofrece crear Layaouts o plantillas de diseño de la aplicación web , dónde se puede definir toda la estructura de la aplicación, como puede ser la cabecera, el contenido y el pie de página.

Por otro lado, también ofrece las funciones necesarias para recuperar los parámetros que estén incluidos en la URL y que se pueden utilizar para recuperar información específica relacionada con la ruta.

Para realizar las demostración se necesitará el package iron:router que se instalará como en los casos anteriores:

meteor add iron:router

Una vez añadido el package, se crearán tres plantillas con las que se hará la demostración. Las plantillas son identificadas por el *tag* ***<template>*** y cada plantilla tiene un nombre por el que es identificada.

En la siguiente imagen se puede ver una de las plantillas creada:

```
<template name="autoform_demo">

      <h1> Simple schemas and Autoforms </h1>

   {{> quickForm collection="Students" id="insertStudent" type="insert"}}
</template>
```

Fig. 15 Plantilla creada para la demostración de iron:router

Como se puede ver la plantilla autoform_demo contiene otra plantilla en su interior (quickForm), que como se vio en la demostración anterior permite generar los formularios automáticamente.

Una vez creada las plantillas se realizará el enrutamiento en el cliente de la siguiente forma:

```
Router.route('/', function () {
  this.render('home_page', {
    //data to send to the template
  });
});

Router.route('/autoform', function () {
  this.render('autoform_demo', {
    //data to send to the template
    });
});

Router.route('/contact', function () {
  this.render('contact_page', {
    //data to send to the template
    });
});
```

Fig. 16 Routing para las tres plantillas creadas

En la cual cada ruta es definida por la URL deseada y el nombre de la plantilla de la cual se quiera mostrar para esa ruta. Cómo se puede observar no es necesario conocer la ubicación exacta de la plantilla, ni hace falta importarla, lo único necesario es saber el nombre de la plantilla que se desee mostrar.

Por ejemplo, para la ruta **contact** la URL definida es **/contact** y la plantilla para mostrar está incluida en la función **render** que en este caso es la plantilla **contact_page.**

Como parámetro opcional, se puede enviar datos a la plantilla y que podrían ser accedidos cómo si fueran variables globales, este punto se verá más adelante.

En el caso de querer acceder a la plantilla autoform_demo, la ruta seria domain.com/autoform y mostraría toda la información que contenga la plantilla autoform_demo.

3.2.3 Utilización de Layaouts

La utilización de Layaouts o plantillas para el diseño de la aplicación web es importante, ya que ayuda al programador a tener una mejor organización. Normalmente una aplicación web se puede dividir en tres partes: la cabecera, el contenido y el pie de página.

La cabecera normalmente contiene la barra de navegación, el logo u otra información importante que se desee mostrar al principio de la página web. Habitualmente la cabecera permanece estática independientemente de la página visitada por el usuario. Respecto al pie de página es similar ya que la información que contiene también permanece estática.

Por lo que la única parte que cambia dependiendo de la ruta o enlace en la que se encuentre es la sección del contenido.

El package iron:router permite trabajar con Layaouts, de esta manera se puede controlar en cada momento que cabecera , contenido y pie de página se desea mostrar para cada ruta.

En esta demostración se utilizará un Layaout llamado **ApplicationLayaout** que contendrá dos secciones, una llamada **nav** que será la cabecera de la aplicación y otra sección llamada **main_content** para mostrar el contenido de cada una de las rutas. Estas secciones se nombrarán como **yields** y serán utilizadas como referencia en el routing.

El diseño HTML y la ubicación de los **yields**, se puede ver en la imagen inferior:

```
<body>

</body>

 <template name='ApplicationLayaout'>

 <div class="row">
    <div class="container">
        <div class="col-md-12">

            {{> yield "nav"}}

        </div>
    </div>
</div>

<div class="row">

  <div class="container">

    <div class="col-md-12">

      {{> yield "main_content"}}

    </div>

  </div>

</div>

 </template>
```

Fig. 17 Diseño HTML del layaout de la aplicación.

Por lo que en la parte superior se tendrá un **yield** con el nombre de **nav**, que es dónde se insertará la barra de navegación que se desee y en la parte

central se tendrá el **yield** main_content que es dónde se mostrara el contenido de la aplicación.

Una vez el diseño de la plantilla de la aplicación esta echa se puede definir una plantilla preestablecida para toda la aplicación y de esta forma tener la misma estructura.

En la siguiente imagen se puede ver la configuración de routing utilizando los yields mencionados anteriormente.

```
Router.configure({

    layoutTemplate: 'ApplicationLayaout'

});

Router.route('/', function(){

    this.render('nav', {
        to: "nav",
        data: { /*data to sent to the template*/ }
    });

    this.render('home_page', {
        to: "main_content",
        data: {/*data to sent to the template*/ }
    });

},
{
    name: 'home'
}

);
```

Fig. 18 Configuración del Layaout

La primera parta de la imagen que hace referencia a **Router.configure**, define cual es el Layaout que será utilizado para la aplicación por defecto. Esta función es opcional ya que para cada ruta se puede elegir un Layaout diferente.

La sintaxis para definir las rutas es similar a los casos anteriores, la única diferencia es el campo ***to*** que hace referencia a los **yields** que se han nombrado anteriormente. Es decir, se define que plantilla será mostrada en cada sección del Layaout.

De esta forma si en la sección **to: main_content** se desea mostrar una plantilla diferente a la de home_page simplemente es cambiar esta plantilla por la plantilla que se desee.

3.2.4 Routing con parámetros dinámicos

El routing con parámetros permite utilizar variables que son proporcionadas en la URL para mostrar información específica. La mejor forma de entender este punto es con el siguiente ejemplo:

Se requiere recuperar la información de un estudiante en concreto dependiendo del ID del estudiante que es proporcionado en la URL.

La URL tendrá el siguiente formato:

/student/id_student

De este modo se necesita recuperar la información de este estudiante en la base de datos utilizando el **ID**. Para realizar este proceso con iron:router, se realiza de la siguiente forma:

Se define una nueva ruta /student y el parámetro viene definido en :_id. En la siguiente imagen se puede apreciar que la nueva ruta está definida como:

/student/:_id

Dónde los dos puntos indican que es un parámetro que está incluido en la URL. El parámetro **_id** es utilizado para recuperar información desde la colección Student con el ID proporcionado en la URL.

```
Router.route('/student/:_id', function(){

    this.render('nav', {
        to: "nav",
        data: { /*data to sent to the template*/ }
    });

    this.render('student', {
        to: "main_content",
        data: function (){
            return Students.findOne({_id:this.params._id});
        }
    });

},
{
    name: 'student'
}

);
```

Fig. 19 Routing con el ID del estudiante

Utilizando una de las plantillas que han sido creadas anteriormente se puede obtener un formulario para editar la información del estudiante que tenga el mismo ID que ha sido proporcionado en la URL.

Finalmente en la siguiente imagen se puede apreciar como en la URL está definido el ID y un formulario con toda la información que pertenece al estudiante por lo que se ha creado una página dinámica gracias a los parámetros proporcionados por el routing a la plantilla.

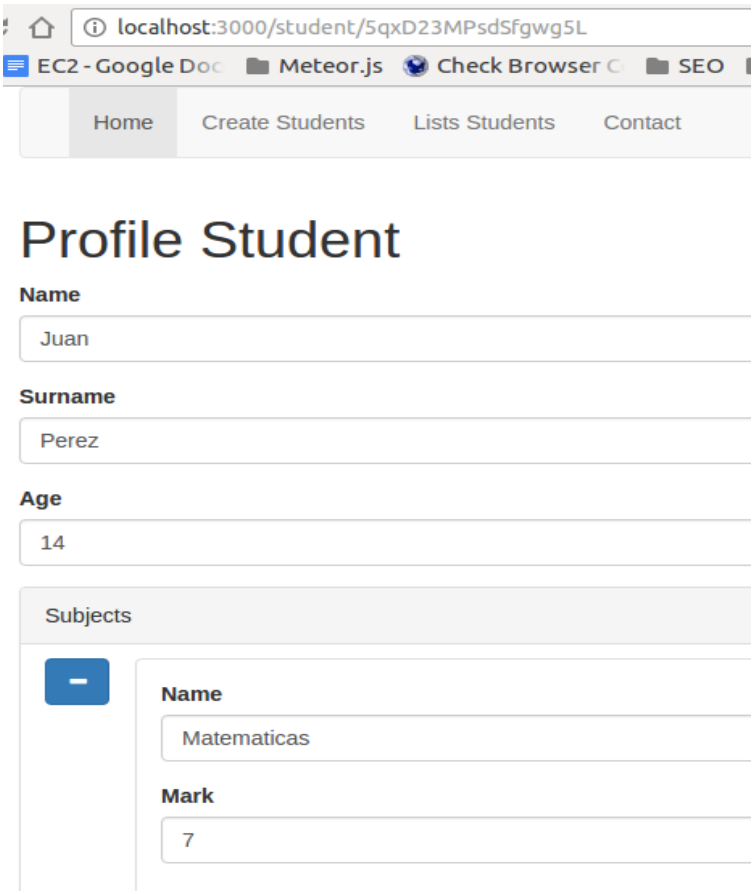

Fig. 20 Formulario para la modificación de un estudiante en concreto

3.2.5 Conclusiones sobre el routing

Al terminar esta demostración se ha podido comprobar que la creación de nuevas páginas es muy simple e independiente a la ubicación de los archivos, a la vez que es flexible ya que permite cambiar el contenido fácilmente para cada una de las rutas. De la misma forma si se desea crear o eliminar una nueva ruta es añadir una función o eliminarla respectivamente.

Por otro lado, la utilización de Layaouts permite un control máximo sobre lo que se desee mostrar para cada una de las rutas y una mejor organización. Otra ventaja de utilizar Layaouts es que facilita el diseño de la aplicación para dispositivos móviles debido a que solo es necesario centrarse en una sola plantilla.

Finalmente, la utilización de parámetros en las rutas lo convierte en una herramienta muy útil que permite al programador recuperar información concreta y entregarla a la plantilla para que sea mostrada.

3.3 Eventos

3.3.1 Introducción a los Eventos

En este apartado se demostrará la flexibilidad a la hora de crear eventos, los cuales permiten actualizar información en la base de datos, variables o ejecutar nuevas acciones. Los cambios realizados en la base de datos o en las variables serán reflejados automáticamente en la aplicación web sin la necesidad de refrescarla o actualizarla, por lo tanto, los clientes estarán viendo los cambios que se realicen en tiempo real.

La declaración de los eventos se realiza de la siguiente forma:

```
"click .change_color": function (event, template) ……
```

Dónde **click** es el evento que ejecuta la acción y **.change_color** es el **target,** es decir la función se ejecuta cuando el evento especificado es realizada sobre el **target** indicado. Para indicar el **target** se utiliza CSS o JQuery **selectors**.

Algunos de esos selectores pueden ser:

Selector	Definición
.red_color	Se seleccionan todos los elementos que contengan la clase red_color.
#red_color	Se selecciona el elemento que contenta el id igual a red_color
p .red color	Se seleccionan todos elementos que contenga la clase red_color y que se encuentren dentro de un párrafo.

Tabla 2 Tabla con los selectores de CSS o JQuery.

3.3.2 Demostración

Para esta demostración se creará una página en la cual se puedan enviar las opiniones de los usuarios sobre la aplicación, es decir una página para recibir feedback. A cada feedback se le podrá asignar un color (tres posibles) y una posición en la lista de feedbacks.

El objetivo es demostrar como los eventos pueden realizar cambios en la base de datos que serán reflejados en tiempo real en el navegador.

La estructura del comentario en la base de datos será la siguiente:

- _id
- text
- color
- position

El diseño de cada comentario está compuesto por el nombre de la persona, el contenido del comentario y un desplegable para seleccionar la posición en la lista de comentarios. Habrá tres posibles colores, dónde cada color tendrá asociado un evento que cambiará el color del comentario. La apariencia final de cada comentario será cómo el de la imagen siguiente:

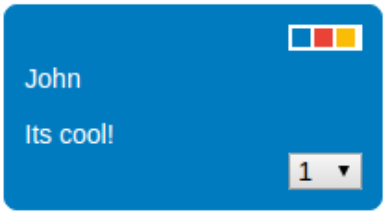

Fig. 21 Ejemplo de un feedback

Para listar los comentarios en la aplicación web, se recuperará la información guardada en la colección feedbacks y que será entregada a la plantilla HTML.

El código que se ha utilizado para enviar la información a la plantilla es el siguiente:

```
this.render('feedbacks', {
    to: "main_content",
    data: { list_feedbacks : Feedbacks.find({}, {sort: {position: 1}})}
});
```

Fig. 22 Lista de feedbacks entregadas a la plantilla

De esta forma para listar los comentarios se utiliza la variable ***lists_feedbacks*** que esta ordenada por la posición del comentario (**sort: {position:1}**). Para recorrer la lista en la plantilla HTML y poder mostrar cada comentario se ha utilizado la función **#each** que permite obtener el nombre, **{{name}}** , el mensaje **{{text}}** y el **{{color}}.**

Al utilizar la función **#each**, no es necesario programar el código HTML para cada uno de los feedbacks ya que todos utilizarán la misma estructura y los únicos cambios serán los que estén guardados en las variables de {{name}}, {{text}} y {{color}}. De esta forma se asegura que cualquier cambio o nuevo comentario siempre sea visible.

Una ventaja importante de la función **#each**, es que cada vez que un evento sea ejecutado dentro de los elementos de la función siempre se podrá

recuperar el **ID** del elemento por lo que siempre se sabrá que elemento de la lista es el que se debe manipular.

En la imagen inferior se puede observar toda la estructura de la plantilla HTML con las variables que han sido mencionadas anteriormente:

```
{{#each list_feedbacks }}

  <div class="col-md-3">

    <div class="message_container" style="background-color: {{color}} ; border-color: {{color}}">

      <div class="choose_colors">
          <span class="color1"></span>
          <span class="color2"></span>
          <span class="color3"></span>
      </div>

      <div class="clear"> </div>

      <div class="c_text">
      <p>{{name}}</p>
      {{text}}

      </div>
```

Fig. 23 Lista de feedbacks

Una vez la plantilla HTML está finalizada se crearán los eventos asociados a cada comentario. Como se ha mencionado anteriormente dentro de la función del evento está disponible el ID del comentario en la base de datos **(this._id)**, por lo que permite realizar cambios en la base de datos exclusivamente en el comentario al cual se le ha cambiado el color o la posición.

De esta forma cada vez que el evento es accionado cambiará el color del comentario y se verá el cambio automáticamente en la página web y lo mismo sucederá si se realiza un cambio en la posición del comentario.

Para realizar el cambio de color en el comentario simplemente es actualizar el color en la base de datos utilizando el **id** del comentario como se puede ver en la siguiente imagen.

```
Template.feedbacks.events({

    //blue
    "click .color1": function (e , t) {


        Feedbacks.update(this._id, {
                $set: { color: "#007bc0" },
        });
    },
```

Fig. 24 Evento para cambiar el color del comentario

Para añadir comentarios se puede utilizar un formulario como en los casos anteriores que introduzca los nuevos mensajes en la colección y de esta forma se conseguiría una página con la lista de comentarios y un formulario para añadirlos. Cómo se ha mencionado anteriormente cada vez que se realice un cambio en los comentarios, estos serán actualizados sin tener que refrescar el navegador por lo que se conseguiría ver los datos en tiempo real. Finalmente, el resultado final sería el siguiente:

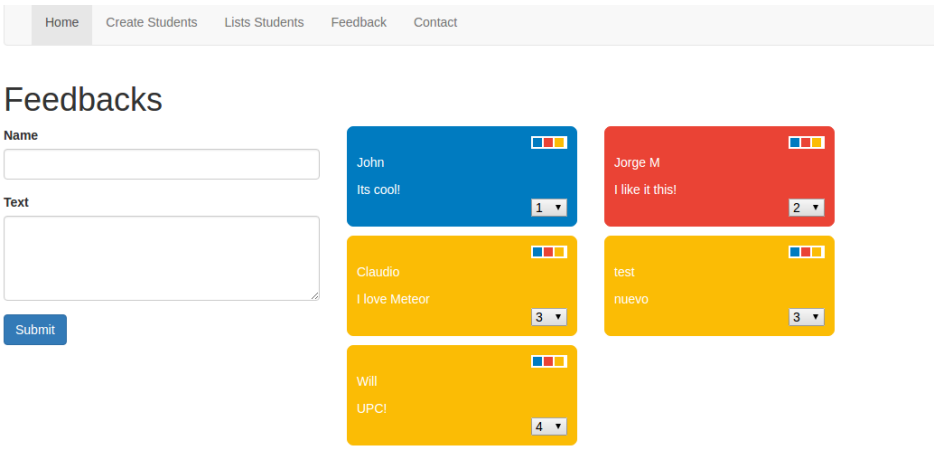

Fig. 25 Lista de Feedbacks con actualización en tiempo real.

En el cual se puede ver la lista de comentarios en tiempo real para todos los clientes que estén visualizando la lista en ese mismo momento, es decir si un comentario es añadido, el color es cambiado o la posición, será reflejado en la aplicación web.

3.3.3 Conclusiones sobre los eventos en MeteorJS

Después de realizar la demostración con eventos se puede ver que la información es actualizada en tiempo real, que en este caso puede ser crear un nuevo comentario, cambiarlo de color y cambiar la posición.

Todas las actualizaciones en la base de datos se han realizado sin la necesidad de crear ningún tipo de llamada AJAX o ninguna conexión con la base de datos lo que permite centrarse en la funcionalidad de la aplicación y no en las conexiones que se ejecutan en segundo plano y son invisibles para el usuario. Cabe destacar que todos los cambios se realizan sin necesidad de refrescar el navegador lo que proporciona una buena experiencia de navegación para el usuario.

3.4 Conclusiones

Al realizar las demostraciones con los packages, routing y eventos se tienen todos los ingredientes para realizar una aplicación web en la que como se ha visto no es necesario centrarse en procesos secundarios como son las comunicaciones con el servidor, la base de datos o como se han visto en las demostraciones ahorrar tiempo en crear formularios o partes de la aplicación web que pueden ser generadas automáticamente.

Referente a los packages en este proyecto solo se han utilizado los más importantes, pero en atmospherejs.com se puede encontrar una gran variedad de funcionalidades desde packages para la gestión de contenidos similar a Wordpress, subir archivos a la aplicación desde la cámara web etc. por lo que las limitaciones son mínimas.

Respecto al uso del routing dinámico, se ha visto que la creación de rutas se realiza de una forma muy fácil en la que una ruta está relacionada con el nombre de la plantilla e independientemente de la ubicación del archivo, lo que permite realizar cambios en las rutas de una forma muy rápida. Además, que la utilización de Layaouts para la aplicación ayudan a tener una estructura clara durante la creación de la aplicación.

Finalmente, la utilización de eventos en MeteorJS ha demostrado que realizar cambios en los registros de la base de datos se realiza de una forma muy rápida en la que solo es necesario saber qué información es la que se desea actualizar y el resto es MeteorJS el que se encarga de actualizar la base de datos y actualizar el contenido en el navegador.

CAPITULO 4. Crear una aplicación web con MeteorJS

4.1 Introducción

Una vez las demostraciones se han realizado, se desarrollará la parte final del proyecto, que consiste en una aplicación web uniendo todas las demostraciones anteriores y con lo que se conseguirá una aplicación con diferentes secciones.

Estas secciones consisten en una página principal que será el Homepage de la aplicación, una sección para el feedback o comentarios, y un menú diferente en el caso de que el usuario este registrado. En el caso de que el usuario este registrado será capaz de crear y editar estudiantes, crear e invitar al resto de estudiantes a una sala de meeting o de chat, o unirse a la sala de meeting a las que haya sido invitado y de esta forma mantener una conversación en tiempo real con los usuarios que estén en línea.

Para implementar el registro de los usuarios se utilizará el package **meteor add accounts-password** que ha sido explicado anteriormente. El cual permite el inicio de sesión o si es necesario el registro de un nuevo usuario que creará automáticamente un estudiante en la base de datos que tendrá la opción de crear un nuevo meeting o ingresar a los meetings que haya sido invitado.

4.2 Fusión de todas las partes

Una vez se tienen todos los packages necesarios el primer paso que se realizará es crear un estudiante cada vez que alguien se registre en la aplicación web.

Para crear el estudiante podemos usar la función **onCreateUser** que es ejecutada cuando un usuario es registrado. Como se puede ver la función recibe como parámetro el usuario que ha sido registrado y será utilizado para crear un nuevo estudiante.

```
Accounts.onCreateUser(function(options, user){

  var e = user.emails[0].address;
  Students.insert( { name: e , surname: e , email: e } );
  return user;

});
```

Fig. 26 Crear estudiante

Una vez el usuario ha sido creado podrá acceder al siguiente menú, ver la imagen inferior, el cual le permite crear y editar estudiantes como en las demostraciones anteriores además de crear meetings o unirse a ellos.

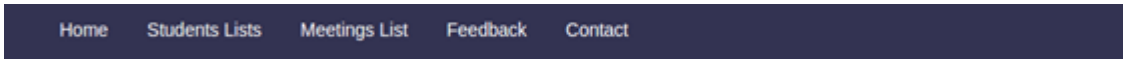

Fig. 27 Menú para el usuario registrado

Cuando el estudiante está en la página de Meetings List, tendrá la opción de crear un meeting con el nombre que desee e invitar a los estudiantes que también estén registrados, como se aprecia en la imagen inferior. En la parte derecha están todos los meetings que el estudiante ha creado o ha sido invitado y por lo tanto puede acceder a ellos.

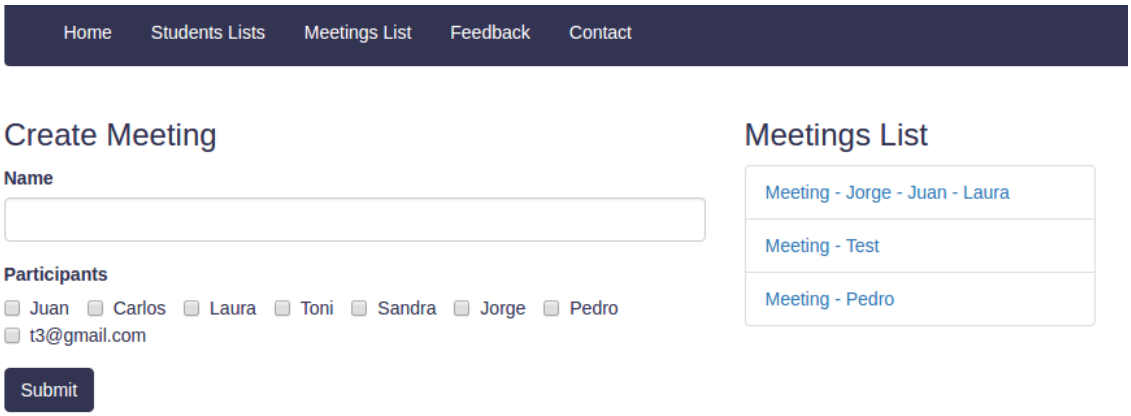

Fig. 28 Página para crear meeting y acceder a ellos

El procedimiento es muy similar a los pasos que se siguieron en las primeras demostraciones. Una vez el estudiante realiza click sobre uno de los meetings que aparecen en la lista de la parte derecha se dirigira a la sala de conversación, en la cual la URL tiene como parametro el ID del meeting, por lo que se puede obtener toda la información relacionada con esta conversación en concreto.

Cuando el usuario este en la sala del meeting podra mantener una conversación con el resto de participantes y podra ver si estan activos o no estan conectados a la aplicación.

Para identificar cuando el usuario ha iniciado sesión se ha utilizado la función **onLogin**, que es ejecutada automáticamente cuando el usuario inicia sesión.

En la siguiente imagen se puede ver con más detalles como se ha realizado este proceso:

```
Accounts.onLogin(function (){
      var u = Meteor.user();
      var e = u.emails[0].address;

      Students.find({email: e}).map(function(s){

          Session.set('student', s._id);
          Session.set('name_student', s.name);

            Students.update(s._id , {
              $set: { status: true },
          });
      });
});
```

Fig. 29 Inicio de sesión del usuario

Una vez el usuario ha iniciado sesión, se realiza una búsqueda en los estudiantes que coincida con el mismo email con el cual ha ingresado para obtener el **ID** del estudiante y realizar la actualización del estado a **true.**
Cuando se cierra sesión el procedimiento es similar pero esta vez el estado es actualizado a **false.** De esta forma obtenemos el estado de cada uno de los estudiantes.

A continuación, es solo necesario poner la condición **{{#if status}},** para identificar el estado y realizar los cambios que se requieran.

En este caso, como se puede ver en la imagen inferior, si el **status** es **true**, se mostrará un pequeño círculo de color verde el cual indicará que el usuario esta online, en caso contrario será de color negro.

```
<ul class="list-inline">
{{#each participants}}
    {{#if status }}
      <li>
            <img src="/img/face1.svg" /> <br> {{name}} -
            <i class="fa fa-circle" aria-hidden="true" style="color:green;"></i>
       </li>

    {{else}}
      <li>
          <img src="/img/face4.svg" /> <br> {{name}} -
          <i class="fa fa-circle" aria-hidden="true"></i>
      </li>
    {{/if}}
{{/each}}
</ul>
```

Fig. 30 Estado de los participantes

Una vez realizada la parte de identificar cuando el usuario esta activo, se desarolló toda la interfaz del usuario para mantener una conversación con los

usuarios que esten en linea. La estructura para mostrar los mensajes es muy simililar a la utilizada en la pagina de feedbacks anteriormente ya que se utiliza la función **#each** para actualizar los mensajes y cambios. La unica diferencia es que en este caso el nombre del usuario es obtenido a traves de la sesion creada cuando el usuario ha iniciado sesion. La interfaz creada para mantener una conversación con los demás usuarios se puede ver en la siguiente imagen:

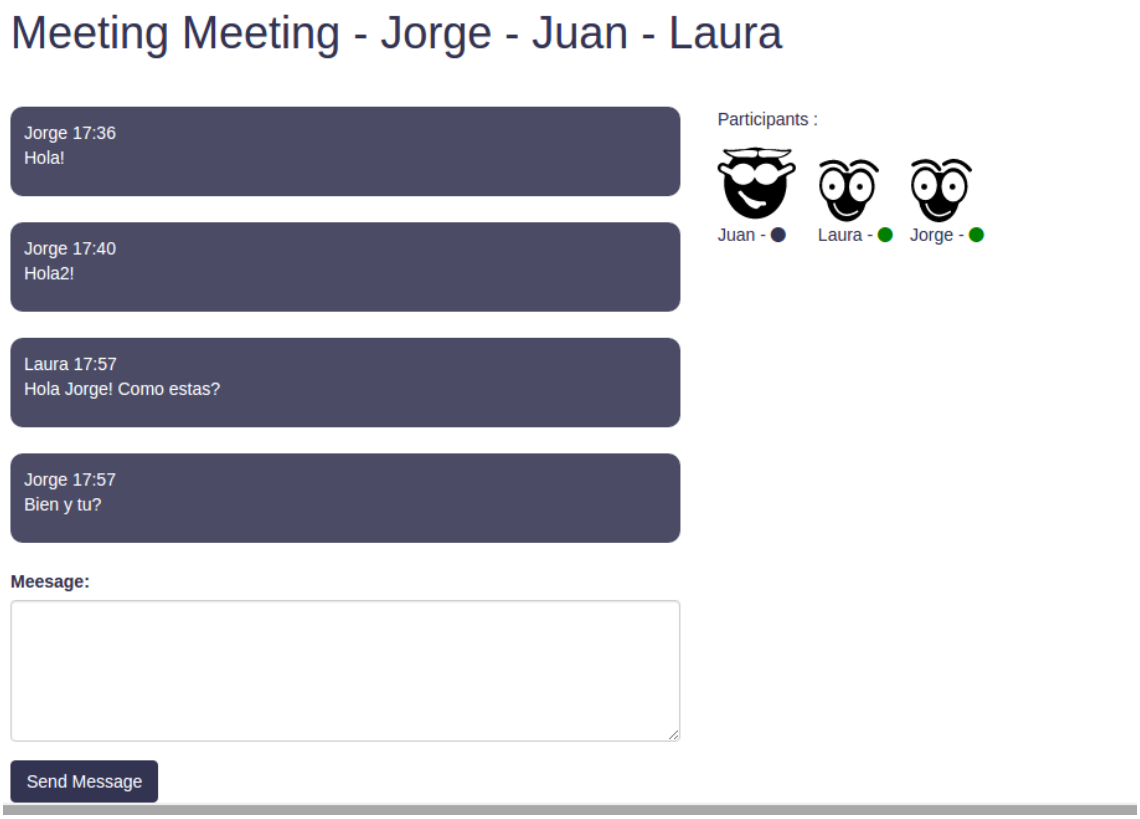

Fig. 31 Sala de Meeting

Donde se puede observar que en la parte izquierda están todos los mensajes que han sido enviados al grupo, el usuario quien lo envió y la hora. Todos los nuevos mensajes se actualizarán automáticamente debido a que son guardados en la base de datos y cualquier cambio relacionado con el meeting será actualizado automáticamente.

En la parte derecha se puede observar los usuarios que están invitados al meeting y el estado de ellos, si el estado es de color verde indica que el estudiante está online y en el caso contrario indicará que el usuario no está conectado a la página web.

Por otro lado, como se ha mencionado anteriormente en el proyecto crear aplicaciones web para diferentes dispositivos se puede realizar de una forma sencilla ya que al utilizar un layaout común para todas las paginas, solo se necesita enfocarse en el diseño de una sola plantilla y no en cada una de las páginas que compone la aplicación.

Cómo se puede apreciar en las siguientes imágenes tomadas en un dispositivo Samsung y un iPad el resultado es muy positivo, ya que ofrece las mismas funciones que en la versión de escritorio:

Fig. 32 Homepage en dispositivo Samsung

Fig. 33 Conversación en un dispositivo Samsung

En la versión del Samsung el menú se adapta automáticamente al dispositivo por lo que todas las opciones siguen estando disponibles, además de que la sala de conversación está totalmente ajustada a la pantalla del dispositivo.

En el caso del iPad el menú sigue siendo el mismo que en la versión de escritorio ya que el tamaño de la pantalla del iPad es mayor por lo que no se requiere comprimirlo como en el caso anterior. La estructura de la conversación es muy similar a la del dispositivo Samsung y es totalmente funcional.

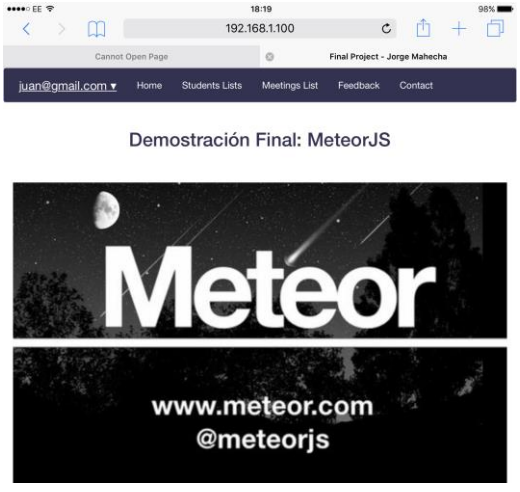

Fig. 34 Homepage en un iPad

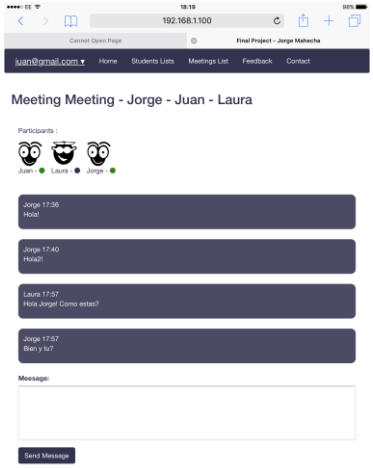

Fig. 35 Conversación en un iPad

El resultado final es una aplicación web en la cual múltiple usuarios pueden acceder a una sala de conversaciones que ha sido creada de una forma muy sencilla y que si se compara con realizarla con lenguajes de programación como pueden ser PHP, el esfuerzo es mucho menor, ya que en PHP hubiera sido necesario crear diferentes funciones para añadir información a la base de datos, crear los formularios manualmente para añadir nuevos meetings o estudiantes y funciones para mantener toda la información en el navegador actualizada. Por lo que la ventaja de utilizar MeteorJS es que todas estas funciones nombradas son automáticamente hechas por el framework.

4.3 Conclusiones

Al realizar esta demostración final, en la que se ha creado una sala de conversaciones totalmente funcional utilizando las demostraciones anteriores y añadiendo algunas partes extras para conseguir el resultado final, se ha demostrado que la creación de una aplicación web en MeteorJS es totalmente posible, además de todos los beneficios que conlleva, ya que el tiempo dedicado es mucho menor que si se hubiera realizado con otras tecnologías clásicas como puede ser PHP debido a la utilización de packages, routing dinámico y la gestión de eventos.

Como se ha mencionado anteriormente, el ahorro de tiempo que conlleva que MeteorJS sea capaz de realizar las constantes actualizaciones entre cliente y servidor da una gran ventaja al programador ya que da la sensación que toda la información está disponible para él y no se tiene que preocupar de cómo conseguirla sino de cómo la quiere utilizar en su aplicación web.

Finalmente, la utilización del routing dinámico permite la flexibilidad de reutilizar plantillas, transferir los datos necesarios y la utilización de parámetros para mostrar información específica, como por ejemplo los datos del estudiante o del meeting. Además, que la utilización de Layaout permite que toda la aplicación tenga la misma estructura por lo que permite que sea mucho más fácil mantenerla y a la misma vez es flexible ya que siempre se pueden cambiar solo las partes necesarias que se necesite sin tener que cambiar cada una de las plantillas.

5. Conclusiones

5.1 Resultados del Proyecto

El resultado final del proyecto ha sido la evaluación del potencial de MeteorJS para la creación de una aplicación web y que ha concluido con el desarrollo de una aplicación que consiste en una sala de conversación para mantener conversaciones con los usuarios que estén en línea. Esta aplicación también incluye funciones como son registrarse e iniciar sesión también crear y modificar los datos del estudiante además de poder acceder a diferentes páginas de la aplicación usando enrutamiento dinámico.

Por lo que se ha demostrado que se puede crear una aplicación web con MeteorJS ya que ofrece todas las herramientas para conseguir este objetivo de una forma sencilla, rápida y con el uso de un solo lenguaje de programación que es JavaScript.

También se ha podido observar que implementar aplicaciones que necesiten actualizar contenido o datos continuamente se puede desarrollar de una forma muy sencilla y rápida, ya que el programador no tiene la preocupación de realizar funciones AJAX para actualizar los datos debido a que se realiza automáticamente con el protocolo **DDP** que se ha nombrado al principio del documento.

Una parte importante del proyecto son los packages ya que permiten utilizar herramientas que ya han sido previamente desarrolladas y de muy fácil integración en los proyectos por lo que la rapidez de trabajo se incrementa considerablemente.

5.2 Inconvenientes

El mayor inconveniente es que es un framework relativamente nuevo por lo que no se sabe el futuro concreto de este framework, debido a que MeteorJS depende de la comunidad, los packages pueden dejar de ser mantenidos por la comunidad o si la lista de nuevos packages es cada vez menor se pondría en peligro la existencia de este framework.

El segundo mayor inconveniente es que al no haber ejemplos claros de empresas utilizando este framework, no se sabe con certeza como funciona en entornos reales donde la demanda es mucho más grande y por lo que el rendimiento pueda ser menor a la misma vez que tampoco se sabe con certeza como trabaja MeteorJS cuando las bases de datos son mucho más grandes.

Otros inconvenientes encontrados al principio del proyecto es que MeteorJS solo soportaba MongoDB como base de datos y Blaze en el frontend por lo que

limitaba el uso del framework en muchos proyectos. A día de hoy esto ya no es una limitación ya que existe Apollo Server que integra GraphQL de Facebook para crear una API con cualquier base de datos y permitir actualizar el contenido en tiempo real. Por otro lado, ya no es necesario utilizar Blaze en el frontend ya que se puede utilizar React.js de una forma muy sencilla y hay bastante documentación en la red.

5.3 Conclusiones Personales

MeteorJS me ha ayudado a adquirir más experiencia en JavaScript y me ha abierto a nuevo mundo de posibilidades que no había conseguido con tecnologías más robustas, pero menos flexibles como son PHP, C# y JAVA. Al adquirir más experiencia con proyectos en Node.js también me ha servido para introducirme en otros proyectos utilizando React.js o Angular.js, por lo que me ha abierto a un nuevo abanico de posibilidades a la hora de crear aplicaciones web.

He aprendido también sobre cómo trabajan las bases de datos basadas en MongoDB y las diferencias con bases de datos relacionales como es MySQL y el rendimiento entre ellas, por lo que he aprendido que MongoDB puede ser más rápido si los datos son guardados con la estructura correcta, pero a la vez también es claro que MySQL es una poderosa herramienta que permite realizar consultas complejas y unir diferentes tablas de una forma sencilla y que MongoDB no permite.

Además, he visto que empresas grandes como son Facebook apuestan fuertemente por este tipo de tecnologías ya que han lanzado frameworks como es React.js y GraphQL que prometen mucho en las nuevas aplicaciones web.

Respecto al mundo laboral, cada vez se buscan más desarrolladores con conocimientos en Node.js, Meteor.js, React.js y GraphQL y están mejor valorados por lo que es un buen indicativo de cómo serán las aplicaciones web del futuro.

5.4 Recomendaciones para un proyecto real

Antes de realizar cualquier proyecto real con MeteorJS es aconsejable ver que frameworks son los que más se adaptarían al proyecto real y ver si MeteorJS es el que más se adecua. Una vez dicho esto se enumeran las siguientes recomendaciones:

- Antes de realizar un proyecto con MeteorJS evaluar cuales son las necesidades que se tienen, ya que si son proyectos muy grandes no es aconsejable utilizar MeteorJS ya que no es un framework tan robusto como sí lo son otras tecnologías.

- No utilizar Blaze en el frontend, ya que en este momento se puede utilizar React.js, del cual hay mucha más documentación y es un proyecto más estable ya que el desarrollador es Facebook.
- Utilizar Apollo Server si se requiere utilizar otro tipo de base de datos diferente a MongoDB.
- Estar constantemente al día en los blogs de MeteorJS para estar al día de las novedades.

CAPITULO 6. Referencias

[1] «DDP,» [En línia] . Disponible: https://meteorhacks.com/introduction-to-ddp/ [Último acceso: febrero 2017].

[2] « aldeed:simple-schemas package,» [En línia] Disponible: https://atmospherejs.com/aldeed/simple-schema . [Último acceso: febrero 2017].

[3] « aldeed:collection2 package » [En línia] Disponible: https://github.com/aldeed/meteor-collection2 [Último acceso: febrero 2017].

[4] « aldeed:autoform package » [En línia] Disponible: https://atmospherejs.com/aldeed/autoform [Último acceso: febrero 2017].

[5] « Accounts-password package » [En línia] Disponible: https://atmospherejs.com/aldeed/autoform [Último acceso: febrero 2017].

[6] « Iron:router package » [En línia] Disponible: https://github.com/iron-meteor/iron-router [Último acceso: febrero 2017].

[7] « Handlebars » [En línia] Disponible: http://handlebarsjs.com/ [Último acceso: febrero 2017].

CAPITULO 7. Glosario

Framework: plataforma de software que permite abstraer al programador de ciertas funciones.

URL: Dirección en la cual se pueden encontrar recursos de la web.

ID: cadena de caracteres que identifica de forma única un elemento de cualquier sistema.

Cliente: Es una aplicación informática que consume un servicio remoto en otro ordenador conocido como servidor.

Servidor: Es una aplicación en ejecución (software) capaz de atender las peticiones de un cliente y devolverle una respuesta en concordancia.

Full-stack: Es un software que proporciona todas las herramientas para trabajar tanto en el cliente como en el servidor.

AJAX: Es una técnica de desarrollo web de aplicaciones que se ejecutan en el cliente, es decir, en el navegador de los usuarios mientras se mantiene la comunicación asíncrona con el servidor en segundo plano

Package: Un módulo de software que realiza una función en particular.

Callbacks: Es una función "A" que se usa como argumento de otra función "B". Cuando se llama a "B", ésta ejecuta "A"

Layaouts: Es una plantilla que sirve para el diseño de la aplicación web.